Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-213Б-23

Студент: Шилов П.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 15.11.24

Москва, 2024

Постановка задачи

Вариант 16.

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Задается радиус окружности. Необходимо с помощью метода Монте-Карло рассчитать её площадь.

Общий метод и алгоритм решения

Использованные системные вызовы:

- int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start)(void *), void *arg)– создание потока
- int pthread_join (pthread_t THREAD_ID, void ** DATA) – ожидание завершения потока
- int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr) – инициализация мьютекса
- int pthread_mutex_lock(pthread_mutex_t *mutex) – блокировка мьютекса
- int pthread_mutex_unlock(pthread_mutex_t *mutex) – разблокировка мьютекса
- int pthread_mutex_destroy(pthread_mutex_t *mutex) – удаление мьютекса

В качестве аргументов командной строки программе радиус окружности, количество точек и количество используемых потоков.

Создаются массив указателей на потоки и массив, каждый элемент которого представляет собой структуру, содержащую необходимые потоку аргументы. Каждому потоку передаются локальное количество точек (points), радиус окружности (radius), указатель на количество точек, попавших внутрь круга (inside_circle) и указатель на мьютекс, который отвечает за добавление счетчика точек, попавших внутрь круга, каждого потока в общий счётчик.

Каждый поток №N отвечает за проверку 1/N от общего количества точек по методу Монте-Карло. В начале итерации поток проверяет каждую из точек, заданных в случайном месте внутри квадрата, описывающего окружность заданного радиуса, на попадание внутрь этой окружности. В конце поток блокирует мьютекс, чтобы добавить локальный счетчик точек внутри круга к общему счетчику точек внутри круга, после чего разблокирует мьютекс.

После завершения всех потоков, программа рассчитывает площадь окружности по формуле и выводит в стандартный поток результат вычислений.

Код программы

main.c

```
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <math.h>
#include <time.h>

#define MAX_THREADS 10
#define BUF_SIZE 1024

typedef struct {
int points;
double radius;
int* inside_circle;
pthread_mutex_t* mutex;
} ThreadArgs;

void* monte_carlo_thread(void* arg) {
ThreadArgs* data = (ThreadArgs*) arg;
int local_count = 0;
for (int i = 0; i < data->points; i++) {
double x = (double)rand() / RAND_MAX * data->radius * 2 - data->radius;
double y = (double)rand() / RAND_MAX * data->radius * 2 - data->radius;
if (x * x + y * y <= data->radius * data->radius) {
local_count++;
}
}

pthread_mutex_lock(data->mutex);
*(data->inside_circle) += local_count;
pthread_mutex_unlock(data->mutex);

return NULL;
}

int double_to_str(double value, char* buffer, int precision) {
int int_part = (int)value;
double frac_part = value - int_part;
char* ptr = buffer;
int n = 0;
if (int_part == 0) {
*ptr++ = '0';
} else {
char temp[20];
char* temp_ptr = temp;
while (int_part > 0) {
*temp_ptr++ = '0' + (int_part % 10);
int_part /= 10;
}
while (temp_ptr != temp) {
```

```
*ptr++ = *--temp_ptr;
n++;
}
}
*ptr++ = '.';

for (int i = 0; i < precision; i++) {
frac_part *= 10;
int digit = (int)frac_part;
*ptr++ = '0' + digit;
frac_part -= digit;
n++;
}
*ptr = '\n';

return n;
}

void print_double(double value) {
char buffer[32];
int n = double_to_str(value, buffer, 6);
write(STDOUT_FILENO, buffer, (n + 2));
}

int main(int argc, char* argv[]) {
if (argc != 4) {
char* msg = "Usage: <radius> <total_points> <num_threads>\n";
write(STDOUT_FILENO, msg, strlen(msg));
return 1;
}

double radius = atof(argv[1]);
int total_points = atoi(argv[2]);
int num_threads = atoi(argv[3]);

if (num_threads <= 0 || num_threads > MAX_THREADS || total_points <= 0 || radius <=
0) {
char* msg = "Invalid input values.\n";
write(STDOUT_FILENO, msg, strlen(msg));
return 1;
}

srand(time(NULL));

pthread_t *threads = (pthread_t*)malloc(num_threads * sizeof(pthread_t));
ThreadArgs *args = (ThreadArgs*)malloc(num_threads * sizeof(ThreadArgs));
int points_per_thread = total_points / num_threads;
int points_in_circle = 0;

pthread_mutex_t mutex;
pthread_mutex_init(&mutex, NULL);

for (int i = 0; i < num_threads; i++) {
args[i].points = points_per_thread;
args[i].radius = radius;
```

```
args[i].inside_circle = &points_in_circle;
args[i].mutex = &mutex;
}

for (int i = 0; i < num_threads; i++) {
pthread_create(threads + i, NULL, monte_carlo_thread, args + i);
}

for (int i = 0; i < num_threads; i++) {
pthread_join(threads[i], NULL);
}

pthread_mutex_destroy(&mutex);

double square_area = 4 * radius * radius;
double circle_area = (double)points_in_circle / total_points * square_area;

print_double(circle_area);

free(threads);
free(args);

return 0;
}
```

Протокол работы программы

Тестирование:

1) $ ./a.out 4 100000000 1 2)$ ./a.out 3 2000000 5 3)$ ./a.out 1 10000000 10

50.253158 28.302659 3.141448

Производительность на примере теста 1:

Количество потоков	Среднее время выполнения, с
1	3,22
5	0,726
10	0,57

Strace:

```
execve("./a.out", ["./a.out", "4", "1000000", "6"], 0x7ffd91cfb608 /* 70 vars */) = 0
brk(NULL)                               = 0x556876aa8000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcced22b80) = -1 EINVAL (Invalid argument)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fd4bff1b000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=58575, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 58575, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd4bff0c000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\302\211\332Pq\2439\235\350\223\322\257\201\326\243\f"...,
68, 896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd4bfc00000
mprotect(0x7fd4bfc28000, 2023424, PROT_NONE) = 0
mmap(0x7fd4bfc28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x28000) = 0x7fd4bfc28000
mmap(0x7fd4bfdbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7fd4bfdbd000
mmap(0x7fd4bfe16000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_DENYWRITE, 3, 0x215000) = 0x7fd4bfe16000
mmap(0x7fd4bfe1c000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|
MAP_ANONYMOUS, -1, 0) = 0x7fd4bfe1c000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fd4bff09000
arch_prctl(ARCH_SET_FS, 0x7fd4bff09740) = 0
set_tid_address(0x7fd4bff09a10)         = 13641
set_robust_list(0x7fd4bff09a20, 24)     = 0
rseq(0x7fd4bff0a0e0, 0x20, 0, 0x53053053) = 0
mprotect(0x7fd4bfe16000, 16384, PROT_READ) = 0
mprotect(0x556875684000, 4096, PROT_READ) = 0
mprotect(0x7fd4bff55000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7fd4bff0c000, 58575)           = 0
getrandom("\xf4\x93\x98\xc1\x37\x35\xe8\xac", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x556876aa8000
brk(0x556876ac9000)                     = 0x556876ac9000
rt_sigaction(SIGRT_1, {sa_handler=0x7fd4bfc91870, sa_mask=[], sa_flags=SA_RESTORER|
SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7fd4bfc42520}, NULL, 8) = 0
rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7fd4bf3ff000
mprotect(0x7fd4bf400000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fd4bfbff910, parent_tid=0x7fd4bfbff910, exit_signal=0, stack=0x7fd4bf3ff000,
stack_size=0x7fff00, tls=0x7fd4bfbff640} => {parent_tid=[13642]}, 88) = 13642
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7fd4bebfe000
mprotect(0x7fd4bebff000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fd4bf3fe910, parent_tid=0x7fd4bf3fe910, exit_signal=0, stack=0x7fd4bebfe000,
stack_size=0x7fff00, tls=0x7fd4bf3fe640} => {parent_tid=[13643]}, 88) = 13643
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7fd4be3fd000
mprotect(0x7fd4be3fe000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
```

```
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fd4bebfd910, parent_tid=0x7fd4bebfd910, exit_signal=0, stack=0x7fd4be3fd000,
stack_size=0x7fff00, tls=0x7fd4bebfd640} => {parent_tid=[13644]}, 88) = 13644
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7fd4bdbfc000
mprotect(0x7fd4bdbfd000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fd4be3fc910, parent_tid=0x7fd4be3fc910, exit_signal=0, stack=0x7fd4bdbfc000,
stack_size=0x7fff00, tls=0x7fd4be3fc640} => {parent_tid=[13645]}, 88) = 13645
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7fd4bd3fb000
mprotect(0x7fd4bd3fc000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fd4bdbfb910, parent_tid=0x7fd4bdbfb910, exit_signal=0, stack=0x7fd4bd3fb000,
stack_size=0x7fff00, tls=0x7fd4bdbfb640} => {parent_tid=[13646]}, 88) = 13646
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7fd4bcbfa000
mprotect(0x7fd4bcbfb000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|
CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fd4bd3fa910, parent_tid=0x7fd4bd3fa910, exit_signal=0, stack=0x7fd4bcbfa000,
stack_size=0x7fff00, tls=0x7fd4bd3fa640} => {parent_tid=[13647]}, 88) = 13647
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x7fd4bfbff910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 13642, NULL,
FUTEX_BITSET_MATCH_ANY) = 0
futex(0x7fd4bebfd910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 13644, NULL,
FUTEX_BITSET_MATCH_ANY) = 0
munmap(0x7fd4bf3ff000, 8392704)         = 0
munmap(0x7fd4bebfe000, 8392704)         = 0
write(1, "50.238399\n", 1050.238399
)           = 10
exit_group(0)                           = ?
+++ exited with 0 +++
```

Вывод

Работа над этой программой потребовала глубокого понимания многопоточного программирования в C. Были реализованы механизмы создания и синхронизации потоков с помощью библиотеки pthread, включая использование мьютексов для безопасного доступа к общим ресурсам. Увеличение числа потоков повысило производительность программы при подсчете точек внутри окружности, что в итоге ускорило процесс вычисления площади окружности. В результате была создана программа, демонстрирующая понимание концепций многопоточного программирования и обработки данных в C.